### 7.3.3 Ejemplo de hoja XSLT

Este primer ejemplo servirá para ilustrar la creación de hoja con XSLT mediante los elementos anteriores, aunque no usaremos muchos de ellos porque no tienen aplicación en una hoja sencilla.

Dado el siguiente esquema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="www.cc.uah.es/jmgm/xml/libros/ejemplos/esquemaSimple"
           xmlns:jmgm="www.cc.uah.es/jmgm/xml/libros/ejemplos/esquemaSimple"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xs:element name="Raiz">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="elem1" type="xs:string"/>
            <xs:element name="elem2" type="xs:string" minOccurs="0"/>
            <xs:element name="elem3" type="xs:string" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

El siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="H:\EjemploSimpleXSLT\hojaSimple.xslt"?>

<Raiz xmlns="www.cc.uah.es/jmgm/xml/libros/ejemplos/esquemaSimple"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="www.cc.uah.es/jmgm/xml/libros/ejemplos/esquemaSimple
H:\EjemploSimpleXSLT\esquemaSimple.xsd">

   <elem1>valor elem1 </elem1>
   <elem2>valor elem2 </elem2>
   <elem3>valor elem3 </elem3>
</Raiz>
```

Y la hoja siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
               omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:value-of select="."/>
```

```
    </xsl:template>
</xsl:stylesheet>
```

El resultado es:

```
valor elem1 valor elem2 valor elem3
```

### 7.3.4 Procesado de una hoja XSLT

Procesar una hoja equivale a procesar una lista de nodos del documento origen para ir creando fragmentos del árbol de salida. La lista de nodos empieza siempre con el nodo raíz. Como procesar un nodo es aplicar la plantilla que lo selecciones, habrá que aplicar una plantilla para el nodo raíz. La aplicación de plantillas generará más nodos para la lista de nodos a procesar y así se irán procesando en orden hasta que la lista esté vacía.

¿Qué ocurriría si existe ninguna plantilla para el nodo seleccionado?

La respuesta es que se aplicaría la plantilla predeterminada. Para los elementos se aplicaría la plantilla siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="*|/">
      <xsl:apply-templates/>
   </xsl:template>
</xsl:stylesheet>
```

Esta plantilla acepta cualquier nodo y pone todos sus hijos, ya sean elementos o texto, en la lista de nodos a procesar. Esto quiere decir que si en algún momento hay algún nodo en la lista de nodos a procesar para el que no hemos definido plantilla, se aplicará esta plantilla y su hijos pasarán a la lista, si tampoco hemos definido reglas para sus hijos, se seguirá el proceso hasta que sean insertados, procesados por defecto y eliminados todos los descendientes de este nodo de la lista.

Para los atributos o nodo de texto, la plantilla predeterminada es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="text() |@*/">
      <xsl:value-of select="."/>
   </xsl:template>

</xsl:stylesheet>
```

Por tanto, nuestra misión será especificar plantillas que procesen los nodos de los que queremos extraer información y modelar la salida generada para esos nodos. Lo normal es que generemos plantillas para el nodo raíz y establezcamos, desde esta plantilla que otros nodos entran en la lista, pero si no lo hacemos, la aplicación automática de plantillas lo hará por nosotros y al ir seleccionando nodos, al final seleccionará nuestras plantillas.

### 7.3.5 Elementos más significativos de XPath

Aunque el Capítulo 9 trata en profundidad XPath, debido a la necesidad e utilizarlo en las expresiones de selección vamos a incluir una tabla resumen del significado de los elementos más relevantes y algunos ejemplo.

| ELEMENTO | SIGNIFICADO | EJEMPLO |
|---|---|---|
| . | nodo actual | <xsl:value-of select=”.”/> |
| hijo | elemento hijo del actual | <xsl:value-of select=”hijo”/> |
| hijo/nieto | elemento nieto del hijo del actual | <xsl:value-of select=”hijo/nieto”/> |
| ./hijo | elemento hijo del actual. Equivalente a hijo | <xsl:value-of select=”./hijo”/> |
| * | todos los nodos hijo del nodo actual. Equivalente a ./*.Se puede combinar con las anteriores. | <xsl:value-of select=”./*”/><br><br><xsl:value-of select=”hijo/*”/> |
| .. | nodo padre del nodo actual | <xsl:value-of select=”..”/> |

| | | |
|---|---|---|
| ../primo/* | posible combinación de elementos. | |
| @at1 | atributo at1 del elemento actual | <xsl:value-of select="@at1"/> |
| ../primo/@at2 | atributo at2 de un nodo primo hijo del padre del nodo actual | <xsl:value-of select="../primo/@at2"/> |
| @* | selección de todos los atributos | <xsl:value-of select="../primo/@*"/> |
| // | todos los descendientes (todos los niveles) del nodo raíz | <xsl:value-of select="//"/> |
| hijo// | todos los descendientes del descendiente hijo del nodo actual | <xsl:value-of select="hijo//"/> |
| //nodoPerdido | todos los nodos llamados nodoPerdido que existan en cualquier parte del árbol. | <xsl:value-of select="//miNodo/*/@at5"/> |
| /primo//nodoTal | elección del nodo primo desde la raíz sin tener el cuenta el nodo actual. Selección de todos sus nodos hijos que se llamen nodoTal sin importar la profundidad a la que estén. | <xsl:value-of select="/primo//elNodo/@at2"/> |
| hijo[@at1='val'] | selección de aquellos nodos hijo que cumplan la condición (su atributo at1 igual a val. Atención a las comillas simples. | <xsl:value-of select="hijo[@at1='5'"/> |

*Tabla 7.1: Resumen XPath*

### 7.3.6 Creación de plantillas

La plantilla más básica, no hace nada con su nodo. Sería una plantilla como:

```
<xsl:template match="nodo">
</xsl:template>
```

Una plantilla algo más interesante producirá algún tipo de salida para su nodo asociado. Esta salida es algún tipo de texto, escrito directamente o calculado en función de valores de los nodos o funciones especiales. Este texto puede ser texto plano o texto con marcado. Por ejemplo:

```
<xsl:template match="nodo">
     Texto asociado al nodo procesado
</xsl:template>


<xsl:template match="nodo">
   <br> Texto asociado al nodo procesado
   </br>
</xsl:template>
```

Sólo con estas posibilidades, se pueden realizar muchas tareas importantes, pero para que las hojas alcancen toda su potencia, necesitamos algo más, necesitamos acceso a los nodos procesados y sus atributos de forma automática, necesitamos repetición y selección condicional de nodos. En definitiva necesitamos poder generar valores calculados para el contenido de salida.

Para esto, existen una serie de instrucciones modeladas como elementos de la especificación de XSLT que veremos, a continuación.

### 7.3.6.1 INSTRUCCIÓN <APPLY-TEMPLATE>

Sirve para llamar a otras plantillas desde el interior de una plantilla. Con esta instrucción añadimos nodos a la lista de procesado. Se añaden los nodos seleccionados mediante el atributo "*select*".

La sintaxis es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0"
             encoding="UTF-8" indent="yes"/>

   <xsl:apply-templates select="Expresion XPath" mode="identificador">

   </xsl:apply-templates>

</xsl:stylesheet>
```

Los atributos que tiene este elemento son:

- select → Mediante una expresión XPath permite seleccionar nodos para incluirlos en la lista de origen.

- mode → Identificativo de la plantilla. Sirve para diferenciar cuando se compara una misma parte del documento XML, pero en cambio cada plantilla realiza una actuación diferente.

Como ejemplo supongamos que se tiene la siguiente DTD:

```
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT Nombres (nombre*)>
```

Un documento XML válido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!—Ejemplo para XSLT -->
<!DOCTYPE Nombres SYSTEM "ejemplo_para_XSLT.dtd">
<Nombres>
        <nombre>XML</nombre>
        <nombre>C++</nombre>
        <nombre>HTML</nombre>
        <nombre>Java</nombre>
        <nombre>Javascript</nombre>
</Nombres>
```

Y un ejemplo de XSLT puede ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
       <html>
         <head>Ejemplo de XSLT </head>
         <body>
           <xsl:apply-templates/>  <!--tomo todos los hijos en la lista-->
         </body>
       </html>
   </xsl:template>
   <xsl:template match="nombre">
      <b>Tema</b>
   </xsl:template>
</xsl:stylesheet>
```

La ejecución de esta instrucción “congela” la generación de salida para el nodo actual hasta que se haya procesado los nodos que se añaden a la lista. Por eso, en el ejemplo anterior se coloca dentro del marcado “*body*”, consiguiendo que la salida del proceso para todos sus nodos hijos se coloque dentro de este marcado.

### 7.3.6.2 INSTRUCCIÓN <XSL:VALUE-OF>

Este elemento busca en el nodo que viene indicado con la expresión XPath del atributo "*select*", e inserta el valor del atributo o elemento encontrado en el árbol de salida.

La sintaxis es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
       <xsl:value-of select="expresion_XPath" disable-output-escaping="yes">
       </xsl:value-of>
   </xsl:template>
</xsl:stylesheet>
```

Los atributos que puede contener son lo siguientes:

- select → Muestra con que debe compararse el árbol del documento XML origen. El valor de este atributo es una expresión XPath.

- disable-output escaping → Hace que en el documento de salida muestre o no los caracteres "&" y "<", en lugar de "&" o "<". Cuando el valor de este atributo es "yes" pone en el documento de salida los caracteres "&" y "<". En cambio si el valor de este atributo es "no" pondrá como salida "&" o "<" respectivamente.

Con la misma DTD del ejemplo anterior se crea el siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Nombres SYSTEM "ejemplo_para_XSLT.dtd">
<Nombres>
        <nombre>&lt;XML&gt;</nombre>
        <nombre>&lt;C++&gt;</nombre>
        <nombre>&lt;HTML&gt;</nombre>
        <nombre>&lt;Java&gt;</nombre>
        <nombre>&lt;Javascript&gt;</nombre>
</Nombres>
```

Una hoja de estilos XSLT podría ser la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
      <xsl:value-of select="/name" disable-output-escaping="no">

      </xsl:value-of>
</xsl:stylesheet>
```

Como resultado se obtendrá el contenido de los elementos "name" con el correspondiente "<" delante y detrás. Es decir, así:

```
&lt;XML&gt; &lt;C++&gt; &lt;HTML&gt; &lt;Java&gt; &lt;Javascript&gt;
```

En cambio esta hoja de estilos:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
      <xsl:value-of select="/name" disable-output-escaping="yes">
      </xsl:value-of>
</xsl:stylesheet>
```

Dará como resultado el contenido de los elementos "*name*" entre los caracteres "<" y ">".

Es decir, la salida será:

```
<XML> <C++> <HTML> <Java> <Javascript>
```

### 7.3.6.3 INSTRUCCIÓN <XSL:ELEMENT>

Se utiliza para insertar elementos de marcado en el árbol del documento destino cuyo nombre y atributos calculamos en base a los nodos del documento origen.

La sintaxis es la siguiente:

```
<xsl:element name="expresion_XPath" namespace="espacio de nombres a usar"
             use-attribute-sets="atributos">
</xsl:element>
```

En la declaración de este elemento se pueden utilizar los atributos siguientes:

- name → Mediante una expresión XPath se hace que el nombre del elemento del documento destino sea dinámico y dependa del documento origen.
- use-attribute-sets → Permite añadir atributos al elemento mediante la referencia a conjuntos de atributos definidos como elementos globales.
- namespace → Para especificar el espacio de nombres al que pertenecerá el elemento.
- xml:space → Indica si debe preservar o no los espacios en blanco.

### 7.3.6.4 INSTRUCCIÓN <XSL:ATTRIBUTE>

Igual que el elemento "<xsl:element>", pero éste sirve para añadir atributos de forma dinámica a un árbol de un documento XML.

```
<xsl:attribute name="Ejemplo" namespace="Espacio_de_Nombres"
               xml:space="default | preserve ">
</xsl:attribute>
```

Los atributos que puede llevar son:

- name → Corresponde con el nombre del atributo que puede ser escrito o calculado con una expresión XPath..
- namespace → Para especificar el espacio de nombres al que pertenecerá el atributo.

- xml:space → Indica si va a preservar o no los espacios en blanco.

### 7.3.6.5 INSTRUCCIÓN <XSL:TEXT>

Este elemento inserta texto (PCDATA) en el documento final de salida. La sintaxis es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:text disable-output-escaping="yes|no"></xsl:text>
 </xsl:stylesheet>
```

Este elemento es útil para introducir espacios en blanco o cualquier otro carácter que de otro modo no se pudiera insertar o visualizar en el documento final de salida.

El atributo que puede acompañar a este elemento es:

- disable-output escaping → Hace que en el documento de salida muestre o no los caracteres "&" y "<" en lugar de "&" o "<". Cuando el valor de este atributo es "yes" pone en el documento de salida los caracteres "&" y "<". Si el valor de este atributo es "no" pondrá como salida "&" o "<" respectivamente.

### 7.3.6.6 INSTRUCCIÓN <XSL:IF>

Es una instrucción de procesamiento condicional, respecto al contenido del atributo "test". Si la comparación es cierta se evalúa el contenido del elemento "<xsl:if>", en caso contrario no se evalúa. La sintaxis es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:if test="condicion"></xsl:if>
</xsl:stylesheet>
```

El atributo que se permite utilizar en este elemento es:

- test → Este atributo indica la expresión lógica con la que se tiene que comparar.

El valor del atributo "test" debe contener los caracteres ">", "<" en lugar de ">" y "<" respectivamente.

### 7.3.6.7 <XSL:CHOOSE>, <XSL:WHEN> Y <XSL:OTHERWISE>

Permite realizar comparaciones más estrictas que el elemento "<xsl:if>" y con más variedad de casos. La sintaxis que utiliza es la siguiente:

```
<xsl:choose>
    <xsl:when    test = "boolean-expression">
              <!-- Contenido: template -->
    </xsl:when>
    <xsl:when    test = "boolean-expression">
              <!-- Contenido: template -->
    </xsl:when>

     ...

    <xsl:otherwise>
              <!-- Contenido: template -->
    </xsl:otherwise>
</xsl:choose>
```

Al igual que los elementos "<xsl:if>", el único atributo que soporta es "test". Aunque con el elemento "<xsl:choose>" está asociado un elemento denominado "<xsl:otherwise>", el cual se evalúa siempre que ninguna de los anteriores haya sido verdadera. Este elemento no es obligatorio.

Existe un caso especial con este elemento. Cuando sean correctas más de una sentencia "<xsl:when>". Cuando esto ocurra únicamente se evaluará la primera que haya sido correcta, saliendo del elemento "<xsl:choose>" a continuación.

El contenido de los atributos "test" deben ser caracteres de escape al igual que el elemento "<xsl:if>", esto es debido a que XML no soporta caracteres como "<". Entonces para evitar problemas se escriben los caracteres de escape.

Ejemplos de esta construcción pueden ser:

```
<xsl:if test="Raiz/elem2">
     <xsl:text>hay elementos tipo 2</xsl:text>
</xsl:if>
```

Y también:

```
<xsl:choose>
   <xsl:when test="Raiz/elem2[@at1=10]">
      <xsl:text> -->at1 de elem2 vale 10
      </xsl:text>
   </xsl:when>
   <xsl:when test="Raiz/elem2[@at1=20]">
      <xsl:text> -->at1 de elem2 vale 20
      </xsl:text>
   </xsl:when>
   <xsl:otherwise>
      <xsl:text> -->Esto sólo si no se cumple nada.
      </xsl:text>
   </xsl:otherwise>
</xsl:choose>
```

### 7.3.6.8 INSTRUCCIÓN <XSL:FOR-EACH>

Se utiliza para procesar de forma iterativa diversos nodos del documento XML origen, estos nodo viene representados por la expresión XPath. La sintaxis se muestra en el siguiente ejemplo:

```
   <xsl:for-each select="expresion_XPath">

   </xsl:for-each>
```

El atributo "*select*" permite introducir una expresión XPath que seleccione varios nodos. Para cada nodo seleccionado por la expresión se realizará lo que se incluya en el interior.

El siguiente ejemplo combina esta instrucción con la anterior:

```
<xsl:for-each select="Raiz/elem2">
   <xsl:choose>
```

```
      <xsl:when test="@at1=10">
         <xsl:text>at1 de elem2 vale 10
         </xsl:text>
      </xsl:when>
      <xsl:when test="@at1=20">
         <xsl:text>at1 de elem2 vale 20
         </xsl:text>
      </xsl:when>
      <xsl:otherwise>
         <xsl:text>Esto sólo si no se cumple nada.
         </xsl:text>
      </xsl:otherwise>
   </xsl:choose>
</xsl:for-each>
```

Mediante esta instrucción podemos añadir en cualquier plantilla el procesamiento de todo un conjunto de nodos distintos de los que la propia plantilla selecciona.

### 7.3.6.9 INSTRUCCIÓN <XSL:COPY-OF>

Se utiliza cuando el documento destino tiene una parte de código igual a la del documento XML origen. Este elemento permite copiar secciones del documento origen al documento destino.

La sintaxis de este elemento es:

```
<xsl:copy-of select="expresion_XPath">
</xsl:copy-of>
```

El atributo "*select*" especifica aquel nodo /nodos que se insertaran directamente en el documento destino.

### 7.3.6.10 INSTRUCCIÓN <XSL:COPY>

Es similar al anterior, pero más potente ya que copia el nodo de contexto, así como los atributos y los hijos de dicho nodo. La sintaxis es la siguiente:

```
<xsl:copy xml:space="preserve | default "
          use-attribute-sets="Conjunto_de_atributos">
</xsl:copy>
```

La diferencia respecto al elemento "*copy-of*" es que únicamente copia el nodo actual sobre el que la plantilla se aplica.

Los atributos que le pueden acompañar son:

- xml:space → Indica si va a preservar o no los espacios en blanco.
- use-attribute-set → Indica si va a contener atributos.

### 7.3.6.11 INSTRUCCIÓN <XSL:SORT>

Permite clasificar un elemento "<xsl:apply-templates>" o "<xsl:for-each>".

La sintaxis de este elemento es la siguiente:

```
<xsl:sort lang="idioma" data-type=" text" order=" ascending | descending"
          case-order="upper-first|lower-first" select="expresion_XPath">
</xsl:sort>
```

Los atributos que acompañan a este elemento son:

- select → Mediante el contenido de este atributo (expresión XPath) elige el elemento o atributo que se quiera ordenar. Si existen dos elementos o atributos idénticos, se mostrarán en el documento destino en el mismo orden en que aparecieron en el documento origen.
- data-type → Indica que los datos a ordenar van a ser numéricos o de texto. El valor por defecto de este atributo es texto. Si el valor es "text" significa que ordenará el contenido como si fuera texto. Por ejemplo, si se tuviera que ordenar 1, 7, 12, 5, seleccionando este atributo con "text" el resultado final sería 1, 12, 5, 7; mientras que si fuera numérico el resultado final sería 1, 5, 7, 12.
- order → Indica si el resultado final de la ordenación va a ser ascendente o descendente.
- case-order → Indica si en caso de aparecer mayúsculas junto a minúsculas, el orden en que deberían aparecer. Si el contenido de

este atributo es “upper-first” las mayúsculas irán antes que las minúsculas. Por ejemplo: se tiene la secuencia “adAcDFf”, si se escoge la opción “upper-case” el orden final será “AacdDFf “; mientras que si la opción seleccionada es “lower-first” el orden final será “aAcdDfF”.

#### 7.3.6.12 OTRAS INSTRUCCIONES

Existen unas pocas instrucciones más que se pueden usar dentro de plantillas. Consulte la especificación de XSLT para saber más sobre ellas.

Las instrucciones restantes son:

<xsl:apply-imports>, <xsl:comment>, <xsl:fallback>, <xsl:message>, <xsl:processing-instruction>, <xsl:variable>, <xsl:with-param>.

### 7.3.7 Ejemplo de plantilla XSLT

El siguiente ejemplo buscaría de forma recursiva en un documento XML todos las personas que están dadas de alta en la biblioteca y los muestra en una tabla junto a otros campos como DNI, etc.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
        <html>
        <head>
                <title>Resultado de la consulta de personas </title>
        </head>
        <body bgcolor="ccccff">
        <br/>
        <h1><center><font color="ff00ff"><blink>
                    <strong>CONSULTA DE USUARIOS</strong></blink></font></center>
</h1>
        <center><img src="../gifs/ojo.gif" alt="imagen"/></center>
        <p>
        <img src="../gifs/barra10.gif" align="center" hsapce="75" alt="imagen"/>
         <br/>
        <center>
         <img src="../gifs/lnk_logo.gif" height="100" width="90" align="left"/>
       </center>
        </p><br/><br/><br/><br/>
        <center>
                <table border="2" bgcolor="orange">
                        <tr>
                                <th><font size="3">IDENTIFICADOR</font></th>
                                <th><font size="3">NOMBRE</font></th>
```

```
                                <th><font size="3">APELLIDOS</font></th>
                                <th><font size="3">DNI</font></th>
                        </tr>
                <xsl:for-each select="Base_de_datos/personas ">
                        <tr>
                                <th><xsl:value-of select="id"/></th>
                                <th><xsl:value-of select="nombre"/></th>
                                <th><xsl:value-of select="apellido1"/>
                                    <xsl:value-of select="apellido2"/></th>
                                <th><xsl:value-of select="dni"/></th>
                        </tr>
                        <tr>
                                <th colspan="5">
                                  <center><img alt="imagen" src="../gifs/ojo.gif"/>
                                  </center></th>
                                        <br/>
                                </tr>
        </xsl:for-each>
                </table>
        </center>


    </body>
    </html>
   </xsl:template>

</xsl:stylesheet>
```

Como resultado final, se obtiene la siguiente página HTML:


*Figura 7.3: Resultado final al aplicar la plantilla XSLT.*